OPIC
OFFICE DE LA PROPRIÉTÉ INTELLECTUELLE DU CANADA

CIPO
CANADIAN INTELLECTUAL PROPERTY OFFICE

Ottawa Hull K1A 0C9

(21) (A1) 2,136,155
(22) 1994/11/18
(43) 1996/05/19

(51) Int.Cl.[5] G06F 15/403; G06F 15/72

(19)(CA) **APPLICATION FOR CANADIAN PATENT** (12)

(54) User Interface for Browsing Multidimensional Objects

(72) Ameline, Ian - Canada ;
Borenstein, Howard - Canada ;
Lansche, Martin R. - Canada ;

(71) IBM Canada Limited - IBM Canada Limitée - Canada ;

(57) 18 Claims

CA9-94-030

USER INTERFACE FOR BROWSING MULTIDIMENSIONAL OBJECTS

Abstract

There is provided a graphical user interface and computer-implemented method for browsing a database or like collection of information where data is organized in multiple dimensions. The interface and method operate to display on a computer screen a frame window containing a first listing of objects in the collection, where the objects are grouped in the listing in tree-view form according to a first expansion sequence, and to display in the same window a second similar listing of objects in the collection which replaces the first listing and whose objects are grouped according to a user selectable other expansion sequence. The objects in the first and second listings are each selectable to display further objects comprised in the collection and associated with the selectable objects, the further objects being represented in nested listings immediately following the selectable objects to which the further objects are associated. The interface and method offer multiple concurrent views of the data to be browsed but do not suffer the disadvantage of using a filtering scheme for limiting the amount of information displayed.

USER INTERFACE FOR BROWSING MULTIDIMENSIONAL OBJECTS

Field of the Invention

The present invention relates generally to a graphical user interface and a computer-implemented method for browsing a database or like collection of information, where data is organized in multiple dimensions. More particularly, the present invention relates to a class browser for an object oriented development environment for viewing the contents of a collection of objects, such as a C++ language class library.

Background of the Invention

a) Introduction

Class browsers are useful tools for viewing and manipulating data structures and hierarchies in object oriented development environments. Object Oriented Programming (OOP) is a programming technique whereby software is structured into individual blocks of code known as objects that each define a set of procedures to be performed in response to messages being sent to and received by the object. Object Oriented Programming techniques have evolved from earlier programming models.

b) Traditional Programming Methodologies

Until recently, programmers employed procedural programming techniques which made use of well-known sequential languages such as FORTRAN, COBOL, BASIC, Pascal and C. With such languages, programs can be developed which provide solutions on the basis of a defined sequence of operations. Other traditional programming methodologies that have been less prevalent than procedural methods are logic and functional programming methods. Logic programming techniques are rule-based methodologies which are particularly useful in artificial intelligence applications. The PROLOG language can be said to employ a logic programming methodology. In the functional programming methods, a technique of data abstraction is used to define various data types and functions. One relatively well-known computer language employing a functional methodology is the LISP language, advantageously used today in graphics applications and geographic information systems.

The main problem with traditional programming methodologies is that they tend to be limited in terms of the availability of given data structures and known instruction sets. The programmer is therefore typically faced with formulating the steps of a solution to fit a limited set of constructs. As well, another problem of the traditional programming methodologies is that the code generated as a result of these techniques is not readily re-usable nor easily prone to enhancement, extension or modification. Lastly, traditional programming methodologies have tended to make problem solving rather non-intuitive. OOP programming techniques have sought to address the foregoing problems associated with traditional programming methods.

c) **Object-Oriented Programming Methodologies**

The emergence of OOP programming languages such as C++, Smalltalk, Objective C and other object oriented languages has resulted in a marked shift in programming techniques and requirements. While programmers using traditional programming languages as referred to above had to deal with large collections of modules, data types or functions grouped together into collections of packaged code known as Application Programming Interfaces (API's), programmers making use of OOP methodologies now deal with large and sometimes complex collections of interrelated classes found in class libraries. In the object oriented programming methodology, the programmer can easily extend the properties of existing data types by adding new properties to them while retaining all the old properties from the original type. This particular concept of extending existing data types is termed "inheritance" and instead of the term data type, the word "class" is typically used. The operations of a class are called its members, methods, functions or "member functions". The data fields embodied within a class are frequently called its "attributes" or its "data members". A collection of classes is known as a "class library" and class libraries may either be extended from a single base class, or from a collection of unrelated classes or unrelated sets of extended classes.

d) The C++ Language

The C++ programming language implements the Object Oriented Programming approach. This language borrows most of the syntax and semantics from the C programming language, which is a procedural language, but adds such concepts as accessibility and class members to implement the object oriented methodologies. In the C++ languages, class members may be defined according to various attributes such as virtual, constant, static or volatile. These attributes allow the implementation of various OOP concepts. Class members may be organized along a dimension which is based on the specialization of the member by type, and these member types may be identified as constructors, destructors, functions, variables and types. Class members may also be distinguished on the basis of their levels of access, which may be identified as public, protected and private.

e) Class Browsers

A computer program can be used to view and interrogate the contents of databases and other collections of data or objects using a graphical user interface (GUI). Such a computer program may be termed a "browser". In some applications, a database may consist of facts which can be sorted or organized along a multiple of independent or dependent criteria or dimensions. Where the contents of the database are large, the search procedures are typically accomplished using a command-level database query language, or using different graphical or non-graphical views of the data where each view is typically ordered according to a single dimension of the data.

The internal structures and features of a class or the external relationships between various classes in a class library can be represented by means of a computer program known as a program representation database. The program representation database stores various items of information about the classes, such as their member functions, data members, inheritance interrelationships and levels of accessibility. The program to view and interrogate this data is known as a class browser. The class browser therefore allows the OOP programmer to understand large collections of classes in a class library.

Current class browsers use one of several techniques to display the members of a class in a library of classes. The most pervasive technique is to perform a query against the database which contains the

is to perform a query against the database which contains the representation of class data, and to display all of the results in an ordinary list. Associated with such a list is a set of filters, or icons in the form of binary switches, which can be set on or off with a pointing device or the like. Filters may be defined for different aspects of class members, such as their levels of accessibility, virtualness or other attributes. For instance, the user may filter a query for public members and the results of the search will only create a list of any such public members. The user may also combine two or more filters for a given query. For instance, the user may filter for public members and for constructors. In such a case, only public constructors will be seen in the results list. One example of a browser which utilizes this approach is that taught in U.S. Patent No. 5,339,443, issued August 16, 1994 in the name of Frid-Nielsen and assigned to Borland International Inc.

Some known browsers perform new queries every time a filter is changed. This can result in a very slow query process. Other prior art browsers cache the complete results of a query and only visually filter the results. In this manner a faster query response is obtained. At times, the filtering process is implemented as a reverse filter in which any results which match a given filter are removed from the results displayed for the viewer. Other known browsers may not employ any filtering procedure. However, for complex class libraries, the user may be overwhelmed by the results that would be displayed with a filterless browser.

The problem with employing filtering as a display technique for a class browser is that filters will generally remove all clues to useful information that cannot be revealed on account of the filter settings. For instance, with reference to the example discussed above, if the user has set a public filter together with a constructor filter in order to obtain a listing of public constructors, the only way to find out if there are any protected data members in the class library being queried is to remove the public filter, set the protected filter, remove the constructor filter and then set the data member filter. This results in multiple queries to see all of the members of a class that one is interested in. Typically, the new query information would replace the existing query results listing. Other prior art browsers allow multiple open windows, each having their own filtered view of the class library data. This

permits previous query results to be retained on the screen display. However, the technique still requires multiple queries to obtain certain class information and causes the user the inconvenience of having to manage various windows to view the information he or she is interested in.

Another problem which results from the use of filtering is that some prior art browsers do not show all of the members that a class may inherit from its base classes. This gives a false representation of a class, because it does not give a full list of all the members available in that class. Again, a user would have to resort to multiple queries to arrive at such information when using some of the known prior art browsers.

Attempts have been made heretofore to use a simple tree view structure in class browsers. A well-know example of such a tree view is a file manager found on some common computer operating systems that uses icons, similar in appearance to paper folders, which may be opened to view their contents by means of an input device such as a mouse. A common aspect of a tree control is the ability to expand a node in the tree structure to reveal elements associated with that node. Tree controls use different open/close icons to indicate to the user that a particular node is expandable or collapsible. By way of example, the OS/2® operating system, available from International Business Machines Corporation, of Armonk, New York, uses a graphics programming interface known as Presentation Manager®. This interface employs a container window that implements a tree control which makes use of a graphics element in the form of a pushbutton icon. The icon is represented as a pushbutton labelled "+" in association with every element in the container listing which may be expanded. Once the node is expanded by selecting the "+" icon, the same icon is relabelled "-" to indicate that the node is thereafter collapsible.

Controls have been used to date in browsers to expand down a single dimension of a hierarchy along a fixed expansion sequence. One such browser relates to an object oriented factory management system and is disclosed in U.S. Patent No. 5,295,242 issued March 15, 1994 in the name of Mashruwala et al. and assigned to Consilium Inc. The expansion mechanism in the browser taught by Mashruwala et al. allows each expandable node to be listed in a separate window from the root nodes of the tree structure and each subnode is likewise itself expandable in a separate window than the one in which its parent node resides.

Thus, current class browsers are problematic in using ineffective filtering techniques to display class information, or are otherwise ill-suited to displaying information which is organized along multiple dimensions.

Objects of the Invention

It is one object of the present invention to provide a class browser which offers multiple concurrent views of the data to be browsed but does not suffer the disadvantages of utilizing a filtering scheme for limiting the amount of information displayed.

It is another aspect of the present invention to present a class browser which is suited to viewing multidimensional data and whereby alternative expansion sequences of the data dimensions are made available to the user.

It is a further object of the present invention to provide a class browser wherein all possible information about a class is available by means of an initially collapsed tree-view structure, whereby the user can display any grouping of information via exploration and expansion of selected nodes in the tree, thereby permitting display of all needed information in a single window.

These and other objects and advantages of the present invention are apparent from the detailed description of its preferred embodiments which follows.

Summary of the Invention

According to one broad aspect of the present invention, there is provided a method of operating a computer system for viewing the contents of a collection of objects having multiple dimensions, said computer system including a display screen and a graphical user interface, said method comprising the computer implemented steps of:

(a) displaying on said screen a frame window containing a first listing of objects comprised in said collection, the objects being grouped within said first listing in tree-view form according to a first expansion sequence thereof; and

(b) displaying in said same frame window a second listing of

the objects being grouped within said second listing interactively in tree-view form according to a user selectable other expansion sequence thereof;

wherein objects grouped within said first and second listings are each user selectable to cause interactive display of further objects comprised in said collection and associated with each of said user selectable objects, the further objects being represented in nested listings within said first and second listings immediately following the user selectable objects to which said further objects are associated.

According to another broad aspect of the present invention, there is provided a user interface for browsing the contents of a collection of objects having multiple dimensions, said interface comprising:

(a) means for displaying, on a display screen of a computer system having a graphical user interface, a frame window containing a first listing of objects comprised in said collection, the objects being grouped within said first listing in tree-view form according to a first expansion sequence thereof;

(b) means for displaying in said same frame window, a second listing of objects comprised in said collection replacing said first listing thereof, the objects being grouped within said second listing interactively in tree-view form according to a user selectable other expansion sequence thereof;

(c) means for selecting said user selectable other expansion sequence;

(d) means for selecting objects grouped within said first and second listing to cause interactive display of further objects comprised in said collection and associated with each of said selectable objects, the further objects being represented in nested listings within said first and second listings immediately following the selectable objects to which said further objects are associated.

Brief Description of the Drawings

The present invention is illustrated by way of example, and not of limitation, with reference to the accompanying drawings in which:

Figure 1 is a screen capture of a browser user interface according to the present invention, displaying a frame window that contains a first top-level listing of classes contained in a library, the classes being ordered within the listing according to class inheritance;

Figure 2 is a screen capture of the user interface according to Figure 1, whereby the top-level listing of objects contains members being identified as having multiple instances;

Figure 3 is a screen capture of the user interface according to Figure 1, wherein two members in the first top-level listing of objects have been selected to display further objects comprised in the library and associated with each of the selected objects at a first level of expansion of the listing;

Figure 4 is a screen capture of the interface according to Figure 1, wherein one of the members in the first top level listing of objects has been selected by the user to cause interactive display of a nested listing of further objects associated with the selected object, and wherein an object in the nested listing of further objects has also been selected to show a second level of expansion for the listing;

Figure 5 is a screen capture of the interface according to Figure 1, wherein an object in the second level nested listing shown in Figure 4 has been selected to reveal further objects at a third level of expansion of such further objects constituting functions for a top level class member;

Figure 6 is a screen capture of the interface according to Figure 1, wherein a different second level nested object has been selected for expansion;

Figure 7 is a screen capture of the interface according to Figure 1, showing a change order pull-down menu by which the user can select another expansion sequence for the listing of objects contained in the frame window;

Figure 8 is a screen capture of the interface according to Figure 1, wherein another top level listing of objects comprised in the library has been displayed in the same frame window;

Figure 9 is a screen capture of the user interface of Figure 1,

showing a nested listing of further objects at a first level of expansion associated with one of the objects in the new top level listing of Figure 8;

Figure 10 is a screen capture of the interface of Figure 1, wherein one of the objects in the first level nested listing of Figure 9 has been selected to display further objects associated with the selected object to thereby display a second-level nested listing;

Figure 11 is a screen capture of the interface of Figure 1, whereby an object in the third-level nested listing has been selected to cause display of a third level nested listing associated with the second level object; and

Figures 12 to 14 constitute a high-level flowchart for an algorithm with implements the browser interface of Figures 1 to 11.

Detailed Description of the Preferred Embodiments

The preferred embodiment of the present invention illustrated with reference to Figures 1 to 14 is a browser interface for objects and classes written in the C++ programming language. The classes to be browsed can be collected in the form of a global class library or from a single program. In either case, the term "class library" will be applied here to describe the collection of objects and classes that can be viewed by way of the preferred embodiment of the user interface according to the present invention. Those skilled in this art will readily appreciate that the browser interface of the present invention may be adapted for use in viewing multidimensional collections of objects of any variety, such as information contained in a database for instance.

In Figure 1, the browser interface 10 as represented on the display screen of a computer system having a graphical user interface consists of a frame window 12 with various frame controls and frame window items. Such controls and items may consist of a title-bar 13 having a title-bar icon 14 and window sizing buttons 16, 18. Frame window 12 may also be provided with a menu bar 20 and information areas 22, 24. Other frame controls and additional frame window items (not identified in Figure 1) may include those of a sizing border for selecting an appropriate frame window size on the display screen and vertical or horizontal scroll bars usually disposed at the frame window borders to cause vertical or horizontal scrolling of

the contents of the frame window. The viewing of the library of objects in the manner to be addressed more specifically below may accomplished by way of the client window area 26. All of the foregoing frame controls and frame-window items are well known to those skilled in the art of graphical user interfaces.

The browser according to the preferred embodiment of the present invention permits the user thereof to generate a first top-level listing 28 of objects comprised in a library, whereby these objects are grouped in the first listing in an initially collapsed tree-view structure that may be expanded according to a default expansion sequence thereof. In the example shown in Figure 1, this first top-level listing 28 consists of a listing comprising a particular class name which is being queried by the user, in this instance the class IPushButton 30, followed by each of the base classes 32 to 37 from which the class being queried is derived. Thus, the objects displayed in the first top-level listing 28 are grouped within the listing according to a default expansion sequence that orders the objects at a top level according to their inheritance information. The default expansion sequence therefore begins with class inheritance information.

The C++ programming language supports the concept of multiple inheritance, whereby a class of objects may inherit directly or indirectly from more than one base class. A given first class, therefore, may inherit from a second class more than once if more than one of its intervening base classes inherits from that second class. Where what is known as virtual inheritance has been defined for the first class, that class will only include a single copy of the second class, whereas if virtual inheritance is not defined as an attribute of the first class, multiple copies of the second class will be included in the first class for each time the first class is derived from it. In Figure 2 of the drawings, the class 42 that is being queried, identified by the name ATextDialog, is displayed in a top-level listing together with its base classes 44. The class 42 has two non-virtual multiple dependencies which are defined in relation to the classes IVBase 46 and IBase 48. As illustrated in Figure 2, the first top level listing resulting from a browser query of class 42 will not display additional labels for each of the classes 46, 48 but will simply flag those classes as having multiple instances as shown by text element 50. In the particular example of Figure 2, the text element

50 denotes that each of the classes 46, 48 have two instances of inheritance within the class 42 being queried, namely the class ATextDialog.

Each of the objects represented in the first top-level listings 28, 45 of Figures 1 and 2 is displayed in association with a graphical element in the form of a pushbutton icon 40. The pushbutton 40 is labelled with a "+" symbol and it is user selectable with the aid of a point-and-click device, such as a mouse, to cause the interactive display of further objects comprised in the collection of objects being queried and associated with the particular object in the top-level listing whose associated pushbutton has been selected. For instance, Figure 3 shows the first top-level listing 28 of Figure 1, wherein the pushbuttons 40 associated with the class names IPushButton 30 and IWindow 35 have been selected by the user for display of further objects associated with these classes 30, 35. In each case, the further objects are represented in nested listings 52, and these nested listings appear immediately following the selected classes 30, 35. Each nested listing contains the first level of the expansion sequence for the top-level listing of Figure 1, which is denoted by the access levels of the class members for the classes 30, 35, namely the labels "public", "protected" and "private" 54 to 56.

Once each of the pushbutton icons 40 is selected by the user to expand one or more of the objects of the top-level listing 28 to produce a nested listing 52 of further objects, the icon 40 displays a "-" symbol in the place of the "+" symbol which is displayed on the pushbutton icon 40 prior to its being selected by a user. Such a selected pushbutton icon is referenced 58 in Figure 3 and is found immediately adjacent to the class names IPushButton 30 and IWindow 35. The selected pushbutton 58 denotes that a nested listing of objects 52 associated with a selected object 30, 35 may be collapsed so as to hide from view the nested listing 52. This is shown for the class name IWindow 35 in Figure 4, where its associated selected pushbutton 58 has been deselected by the user. The deselected pushbutton icon 60 is marked with its original "+" sign as found in association with each of the objects of the top-level listing according to Figure 1.

Figure 4 also shows that any of the further objects 54, 55, 56 of nested listing 52 can be selected by the user to cause interactive display of another nested listing 62, once again by the user selecting one of the

pushbutton icons 40 associated with the objects in listing 52. In the example of Figure 4, object 54 has been selected by the user in the manner described above to produce the nested listing 62 of objects 64 to 67 at a second level of expansion denoting the member types of a class such as constructors/destructors, functions, types and variables. Each of the objects 64 to 67 in nested listing 62 appear within the listing 52 immediately following the user selected object 54 named "public". Selected pushbutton 68 appearing adjacent object 54 is marked with "-" symbol to denote that it may be deselected by the user to cause interactive collapse of nested listing 62 within the listing 52.

Turning now to Figure 5, object 65 denoted "functions" has been selected by the user in the manner described previously to generate yet another nested listing 70, such as that referenced 72, associated with the selected object 65 at a third level of expansion of top-level listing 28. In the particular example of Figure 5, the objects in the nested listing 70 are each function type members of class IPushButton 30 having a public access level. For instance for member 72, its function name "defaultStyle" 74 is displayed in bold font together with its return type "IPushButton::Style" 78 and a parameter list 76 for the function identified as "void". Further information for each class member such as the function defaultStyle 72 is identified by a one-letter icon 80. For functions, the icon 80 may be used to identify attributes for C++ class members such as pure virtual ("P"), virtual ("V"), static ("S") and constant ("C"). For class members which are variables, the only attribute identified with an icon 80 is that of static ("S"). The enumeration attribute ("E") is the sole attribute identified by an icon 80 for class members which are types. All of the different attributes identified in one-letter abbreviations by way of the icons 80 can also be associated with different colours to be more readily discernable to the user. The objects in nested listing 70 are grouped in alphabetical order according to their function names. However, those skilled in this art will appreciate that other methods of grouping the objects of nested listing 70, such as by return type, for instance, may be employed in the alternative. As well, other methods of identifying the numerous attributes of the displayed C++ members will be apparent to persons skilled in the art.

The listing of objects which is contained in frame window 12 in

Figure 5 shows three levels of expansion, such levels being associated with the three objects or labels IPushButton 30, "public" 54 and "function" 65. Because each level of expansion is associated with nested listings 52 (not shown in its entirety in Figure 5), 62 and 70, the top level listing 28 of Figure 1 scrolls out of the view of container frame 12. When this occurs as a result of selecting objects at any expansion level for the display of further nested objects, a vertical scroll bar 82 appears within client area 26 of the container frame 12 to alert the user. Vertical scroll bar 82 may be used in the manner well known in this art to cause the interactive scrolling of the objects contained within frame window 12 so that a user may view all of the remaining objects within the top-level listing 28. Likewise, when the displayed contents of frame window 12 are wider than the window size in the horizontal direction, a horizontal scroll bar (not shown) will appear within the client area 26.

Figure 6 shows the same top-level listing 28 of Figure 1, wherein the object IPushButton 30 has been expanded to generate the nested listing 52 as shown in Figure 3, with a further expansion having been selected for the object "protected" 84. In the particular example shown in Figure 6, the second level of expansion according to member type only shows objects identified as functions 86 as there are no other protected members identified as constructors/destructors, types or variables. Thus, at the second-level of expansion, if there are no members of a specific type, then that specific type is not added to the nested listing for a particular object in a nested listing 52 at the first expansion level. This deletion of objects that are not associated with other objects at a next level of expansion is not implemented for objects within the nested listings 52 at the first expansion level.

Those skilled in this art will appreciate that menu items may be provided within menu bar 20 of frame window 12 in order to expand or collapse all of the objects in the frame window at once, or to expand or to collapse all of the objects within a nested listing under a specific object of interest. This would allow the user to navigate all of the objects in a particular listing quickly and effectively.

As illustrated by Figures 1 to 6 of the drawings, the display of objects within the frame window 12 are grouped within the listing according to the default expansion sequence wherein the objects in the top-level listing may each be expanded at up to three nested levels. The

top-level listing contains the name of the class whose related classes we are viewing, together with the names of all of its base classes. The number of objects displayed at this class level will vary depending upon the number of base classes the particular class of interest has. The first level of expansion within the top-level listing is that of the access levels of the members within a particular class, the access levels being defined as being either public, protected or private. The second level of expansion is according to member type information for the class members, and these may be selected from constructors/destructors, functions, types and variables. Finally, the third level of expansion in the default expansion sequence shown in Figures 1 to 6 groups the member names alphabetically by function name, variable name or type name respectively for member types denoted as functions, variables and types. As explained in greater detail below, the user may generate a second top-level listing of objects comprised in the selection of objects being browsed according to a user selectable other expansion sequence. This allows the user to reorder the display of objects using either the access level as the top-level listing of the expansion sequence or the type level as the top-level within the chosen expansion sequence.

As shown in Figure 7, a drop-down menu 88 may be activated by the user by selecting the "order" option 90 within menu bar 20. Drop-down menu 88 provides three expansion sequences according to class level, access level or type level, class level being the default expansion sequence described by way of Figures 1 to 6. In the expansion sequence selected by the user which is shown in Figure 7 to 11, the top-level listing of objects is organized beginning with access level, and the first, second and third expansion levels for the listing are respectively member type, class inheritance and class members. Those skilled in this art will appreciate that any number of expansion sequences may be provided to the user by way of drop-down menu 88 or other appropriate selection means.

By way of example, a user may be primarily interested in the public functions of a given class library. The user would select "access level" as the desired expansion sequence from drop-down menu 88 to cause interactive display of a second top-level listing of objects within the same frame window 12 according to the newly selected expansion sequence. The second top-level listing 92 generated as a result of such a selection is shown by way of Figure 8, and this second listing replaces the first

listing of objects previously contained in the frame window 12. As best shown in Figure 9, the objects associated with the object "public" 94 of Figure 8 may be displayed in a nested listing 96 by user selection of pushbutton icon 95. When nested listing 96 is displayed within the second top-level listing 92 immediately following the selected object 94, the pushbutton icon 95 displays a "-" symbol as at 98 as explained previously. In the manner already described, Figures 10 and 11 demonstrate how the user may generate further nested listings 100 and 102 by sequentially selecting objects 104 and 106.

Those skilled in this art will appreciate that other features may be provided in conjunction with the browser features described above. For instance, for any object in the listing contained in the frame window, the user may select that object and generate a popup menu of actions that can be performed in relation to that object. As well, a find mechanism may be provided in association with the browser so that the user thereof can easily locate a desired text string within all of the objects within the frame window, whether revealed or not by expansion. In the case where an object is hidden from view, the search mechanism may expand the appropriate node within the listing so that the object found by means of the search is visible to the user.

The browser interface as shown in Figures 1 to 11 may advantageously be implemented by using the container control application programming interface (API) of the OS/2® operating system Presentation Manager® available from International Business Machines Corporation of Armonk, New York. As well, if it is desired to implement the browser of the present invention in C++, the interface to the Presentation Manager® container control may be defined by way of the IContainerControl class that is included in the IBM User Interface Class Library as part of the IBM C++ development environment C Set ++® for OS/2. This particular development environment is also available from International Business Machines Corporation of Armonk, New York. Those skilled in this art will readily appreciate that the present invention may be applied to various platforms, operating systems and compilers, using any other application programming interfaces, library objects or custom coding as a means of implementation.

With reference to Figures 12 to 14, the algorithm for the browser interface according to the present invention begins with the creation of base labels under which the class member items of the browsed collection

of objects will be placed. These base labels for the interface previously described are identified with class name, access level and member type. Since the number of items associated with each base label at the class level changes with each different class of objects that this action is performed on, the base labels are created separately for each of the classes, one class at a time. At decision block 108, the first step in the algorithm is to perform a query as to the expansion order for either of the default or user selected expansion sequences. As explained above, the default expansion sequence for the list of members in any given class library to be browsed is class level first, access level second, and type level third. The user may select any one of a number of other expansion sequences as previously explained.

At decision block 108, if access level is the top level in the expansion sequence, then three access labels (public, protected and private) are created for the top-level frame window listing at action block 110. If access level is the top level and type level is queried at decision block 112 as being the first expansion level in the expansion sequence, then the four member type labels (constructor/destructor, type, functions, variables) are created for each of the three access labels. These members type labels are created at action block 114. Following this, the next step in the algorithm is to proceed to decision block 116 of Figure 12 which will be described in greater detail below.

If, on the other hand, the query at decision block 108 is in the negative, such that access level is not the top level of the expansion sequence, the algorithm proceeds to decision block 118 to query whether member type is the top level. In the affirmative, the four member type labels mentioned above are created for the top-level listing. This is accomplished at action block 120. The next step is to query at decision block 122 as to whether access level is the first expansion level in the expansion sequence. In the affirmative, the three access labels mentioned above are created at action block 124 for each of the four member type labels. Following this step, the algorithm proceeds to decision block 116. In the event that the queries defined by decision blocks 118 and 122 are in the negative, the algorithm also proceeds to decision block 116.

In the portion of the algorithm beginning at decision block 116, each class label in the collection of objects is added to the container, together with all of its member items. At this point in the algorithm,

some of the base labels under which the class member items are to be placed will have already been created. At decision block 116, a query determines whether or not a given class label has been added to the container as yet. If this response is in the affirmative, a counter is incremented at action block 126 for each occurrence of the same class. A counter is maintained for each of the classes in the collection of objects, and the final count for each counter is used to generate text element 50 of Figure 2 which denotes instances of multiple dependencies in the inheritance hierarchy. If the query in decision block 116 is in the negative, such that a given class label has not yet been added to the container, then all remaining labels for the class, access and type are added to the container as represented by action block 128 that summarizes the algorithm of Figure 13 which is described in greater detail below. Once all remaining labels for class, access and type have been added to the container, all of the class member data is added to the container under the appropriate base label for class, access and type. This procedure is represented in Figure 12 by action block 130 that summarizes the algorithm of Figure 14 which is described in greater detail below.

Following the sequence of action blocks 126, 128 and 130, decision block 132 queries as to whether the given class has any base classes. In the affirmative, such that the given class has one or more base classes, each of the base classes is selected in succession as the given class at action block 138. The procedure of the algorithm beginning with decision block 115 is then repeated for each of the base classes in succession. If the query of decision block 132 is in the negative, the classes that appear more than once in the inheritance hierarchy are identified at action block 134 and this information together with the base levels and associated member data are displayed in the container window according to the standing expansion sequence in action block 136. This completes the browser interface algorithm.

With reference now to Figure 13, there is illustrated the high-level flow chart for the algorithm described at action block 128, (Figure 12), namely the addition of remaining labels for class and member type. At decision block 140, the algorithm queries whether class is the top-level of the expansion sequence. If the query is answered in the affirmative, the class label of interest is added at action block 142. The next step is to query at decision block 144 whether access is the first expansion

level. In the affirmative, three access labels are created for the class of interest and four type labels are created for each of the three access labels at action block 146. If, on the other hand, the query at decision block 144 is in the negative, four type labels are created for the class of interest and three access labels are created for each of these type labels at action block 148.

If the query at decision block 140 is in the negative, then at decision block 150 an additional query is made as to whether access is the top level of the expansion sequence. If so, then some base labels will have already been created by previous steps of the algorithm. At decision block 152 a query is made as to whether class is the first expansion level. In the affirmative, class labels are created for each of the three access labels and four type labels for each of the created class labels at action block 154. If not, class labels are created for each type label at action block 156.

If the query made at decision block 150 is in the negative, a further query is made at decision block 158 as to whether class is the first expansion level. If so, class labels are created for each of the four type labels and three access labels for each of the created class labels at action block 160. If not, then at action block 162 class labels are created for each of the access labels.

The algorithm shown in high level format in Figure 14 relates to the operation of action block 130 (Figure 12), namely the addition of class member data to labels which have been created according to a given expansion hierarchy of objects. At action block 164, sets are created for the different types of members for each class of interest. In the user interface model described above, where there are four member types defined and three access levels for each member type, twelve such sets will be created under the operation of decision block 164. The class member data is then associated with the relevant set at action block 166. The class member data may be obtained from a program representation database or other like collection of data that will be known to those who are skilled in this art.

At action block 168, each of the sets is sorted alphabetically by function name, variable name or type name, as the case may be. Then, the items from each of the sorted sets are added to the proper base label with which the class member data is associated, and this operation is

represented at action block 170. Finally, at action block 172, the second expansion level of the created listing is reviewed in order to remove any labels at that level which do not have any other objects listed thereunder.

The browser, according to the present invention, provides many of the advantages of the known browsers which offer multiple concurrent views of the data being viewed, but does not suffer the restrictive effects of a filtering scheme that would limit the amount of information displayed. Instead of restricting a user to choose a query limited by the number of available filters, the browser according to the present invention displays all the possible information about a class at once. All of the objects in the tree-view listing of the present browser are initially collapsed, and via exploration of these objects by expansion, the user can display any grouping of information needed in a single frame window. As explained above, if a user wants to view the browsed data in another way, for instance one that is more meaningful to his or her current investigation, the browser according to the present invention offers alternative expansion sequences according to class names, accessibility and member type. Viewing data by way of combinations of these three dimensions is relatively easy to perform. For instance, it is possible for the user to ask for all member data of the same member type regardless of accessibility, or to ask for all public objects pertaining to a certain member type. As will be appreciated by those skilled in this art, this is not done via a filtering scheme nor via a complicated data base query language, but rather by way of familiar tree structure node expansion and contraction operations. The expansion of the various dimensions in the tree structure allows alternate navigational means of understanding the relationships contained in the data being browsed. This allows the programmer to understand classes in a library or program from many different aspects.

Preferred embodiments of the present invention have been described hereabove by way of example only and not of limitation, such that those skilled in this art will readily appreciate that numerous modifications of detail may be made to the present invention, all coming within its spirit and scope.

The embodiments of the invention in which an exclusive property or privilege is claimed are defined as follows:

1. A method of operating a computer system for viewing the contents of a collection of objects having multiple dimensions, said computer system including a display screen and a graphical user interface, said method comprising the computer implemented steps of:

(a) displaying on said screen a frame window containing a first listing of objects comprised in said collection, the objects being grouped within said first listing in tree-view form according to a first expansion sequence thereof; and

(b) displaying in said same frame window a second listing of objects comprised in said collection replacing said first listing thereof, the objects being grouped within said second listing interactively in tree-view form according to a user selectable other expansion sequence thereof;

wherein objects grouped within said first and second listings are each user selectable to cause interactive display of further objects comprised in said collection and associated with each of said user selectable objects, the further objects being represented in nested listings within said first and second listings immediately following the user selectable objects to which said further objects are associated.

2. The method of operating a computer system according to Claim 1, wherein said collection of objects contains classes thereof and said first expansion sequence has a root level which is an ordered representation of a class of objects within said collection and of every base class thereof, said classes being user selectable to cause said interactive display of said further objects.

3. The method of operating a computer system according to Claim 2, wherein each of said classes contains members thereof and a first expansion level for the root level of said first expansion sequence is access level information associated with each of said members within said classes, said access level information being displayed in said nested listings immediately following said user selectable classes, said access level information being user selectable to cause said interactive display of said further objects.

4. The method of operating a computer system according to Claim 3, wherein said access level information is selected from the group comprising public, protected and private.

5. The method of operating a computer system according to Claim 4, wherein a second expansion level for the root level of said first expansion sequence is member type information associated with each of said members within said classes, said member type information being displayed in said nested listings immediately following said user selectable access level information, said member type information being user selectable to cause said interactive display of said further objects.

6. The method of operating a computer system according to Claim 5, wherein said member type information is selected from the group comprising constructors/destructors, functions, variables and types.

7. The method of operating a computer system according to Claim 6, wherein a third expansion level for the root level of said first expansion sequence is member data for each of said members within said classes, said member data being displayed in said nested listings immediately following said user selectable member type information.

8. The method of operating a computer system according to Claim 7, wherein objects represented in said first and second listings are each displayed in association with a graphical element that is user selectable to cause said interactive display of said further objects.

9. The method of operating a computer system according to Claim 8, wherein all objects comprised in said collection are available for viewing by said interactive display of said further objects.

10. A user interface for browsing the contents of a collection of objects having multiple dimensions, said interface comprising:

a) means for displaying, on a display screen of a computer system having a graphical user interface, a frame window containing a first listing of objects comprised in said collection, the objects being grouped within said first listing in tree-view form according to a first expansion sequence thereof;

b) means for displaying in said same frame window, a second listing of objects comprised in said collection replacing said first listing thereof, the objects being grouped within said second listing interactively in tree-view form according to a user selectable other expansion sequence thereof;

c) means for selecting said user selectable other expansion sequence;

d) means for selecting objects grouped within said first and second listing to cause interactive display of further objects comprised in said collection and associated with each of said selectable objects, the further objects being represented in nested listings within said first and second listings immediately following the selectable objects to which said further objects are associated.

11. The interface according to Claim 10, wherein said collection of objects contains classes thereof and said first expansion sequence has a root level which is an ordered representation of a class of objects within said collection and of every base class thereof, said classes being user selectable to cause said interactive display of said further objects.

12. The interface according to Claim 11, wherein each of said classes contains members thereof and a first expansion level for the root level of said first expansion sequence is access level information associated with each of said members within said classes, said access level information being displayed in said nested listings immediately following said selectable classes, said access level information being selectable to cause said interactive display of said further objects.

13. The interface according to Claim 12, wherein said access level information is selected from the group comprising public, protected and private.

14. The interface according to Claim 13, wherein a second expansion level for the root level of said first expansion sequence is member type information associated with each of said members within said classes, said member type information being displayed in said nested listings immediately following said selectable access level information, said member type information being selectable to cause said interactive display of said further objects.

15. The interface according to Claim 14, wherein said member type information is selected from the group comprising constructors/destructors, functions, variables and types.

16. The interface according to Claim 15, wherein a third expansion level for the root level of said first expansion sequence is member data for each of said members within said classes, said member data being displayed in said nested listings immediately following said selectable member type information.

17. The interface according to Claim 16, wherein the interface further includes a graphical element that is displayed in association with objects represented in said first and second listings, said graphical element being selectable to cause said interactive display of said further objects.

18. The interface according to Claim 17, wherein all objects comprised in said collection are available for viewing by said interactive display of said further objects.

2136155

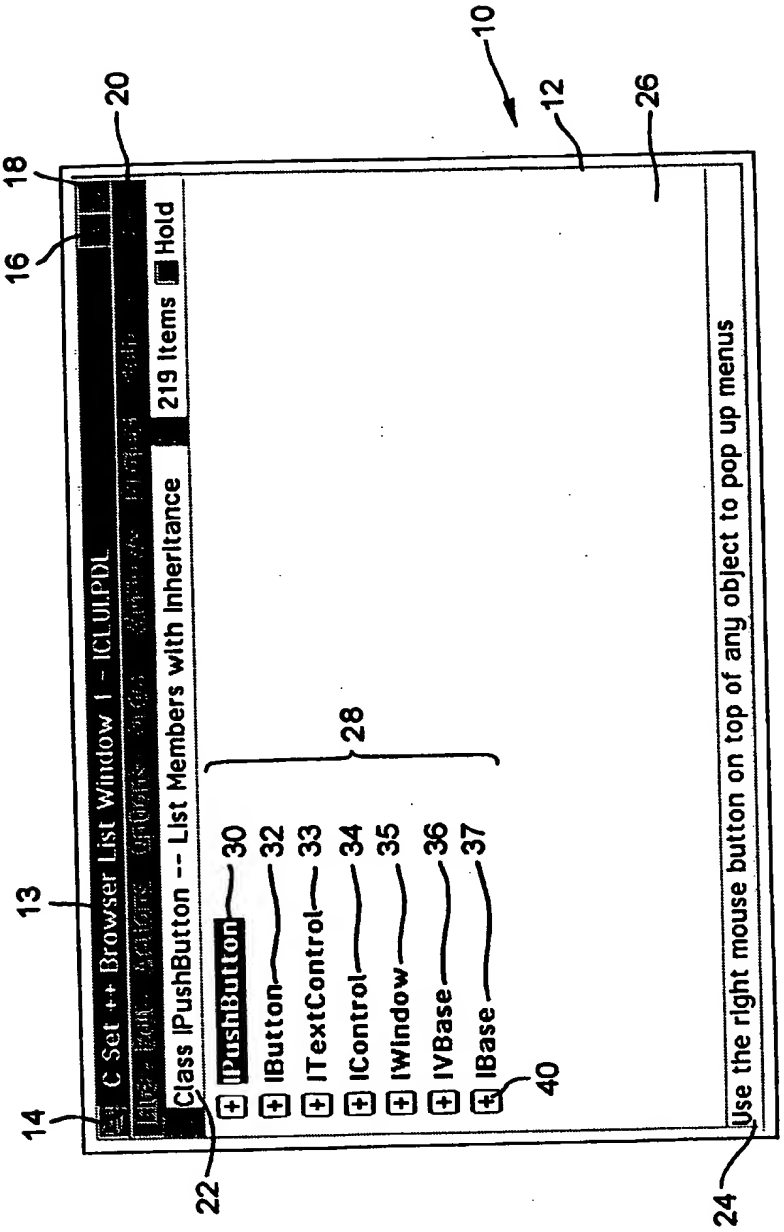

FIG. 1

2136155

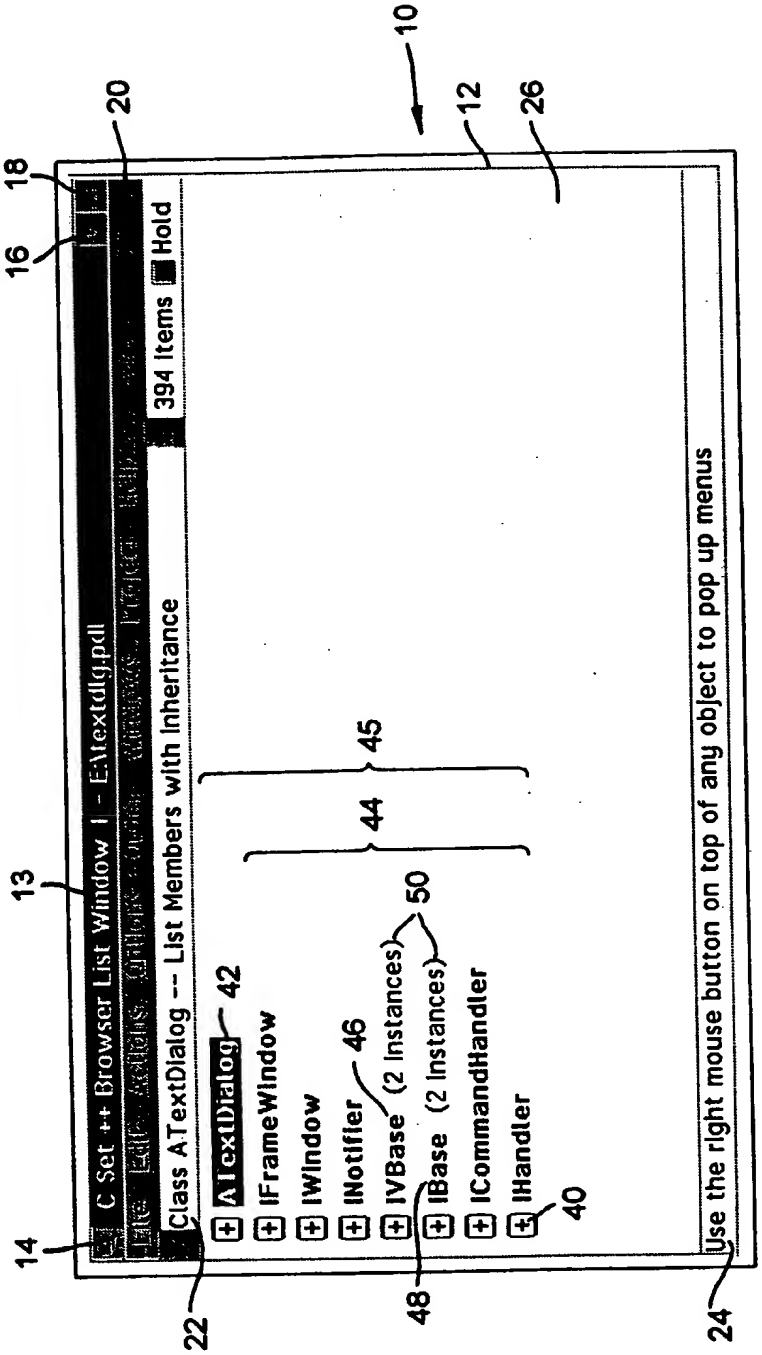

FIG. 2

2136155

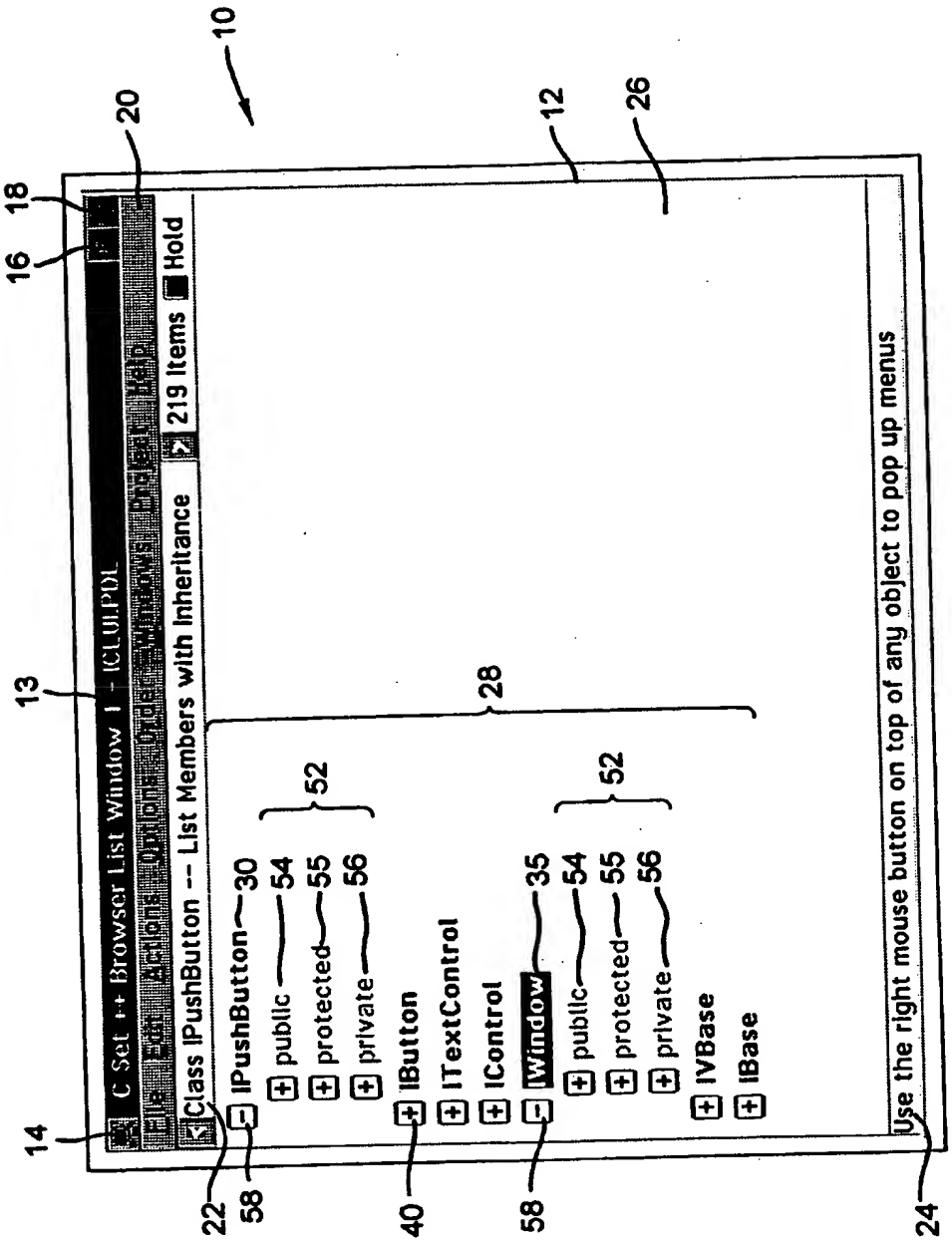

FIG. 3

2136155

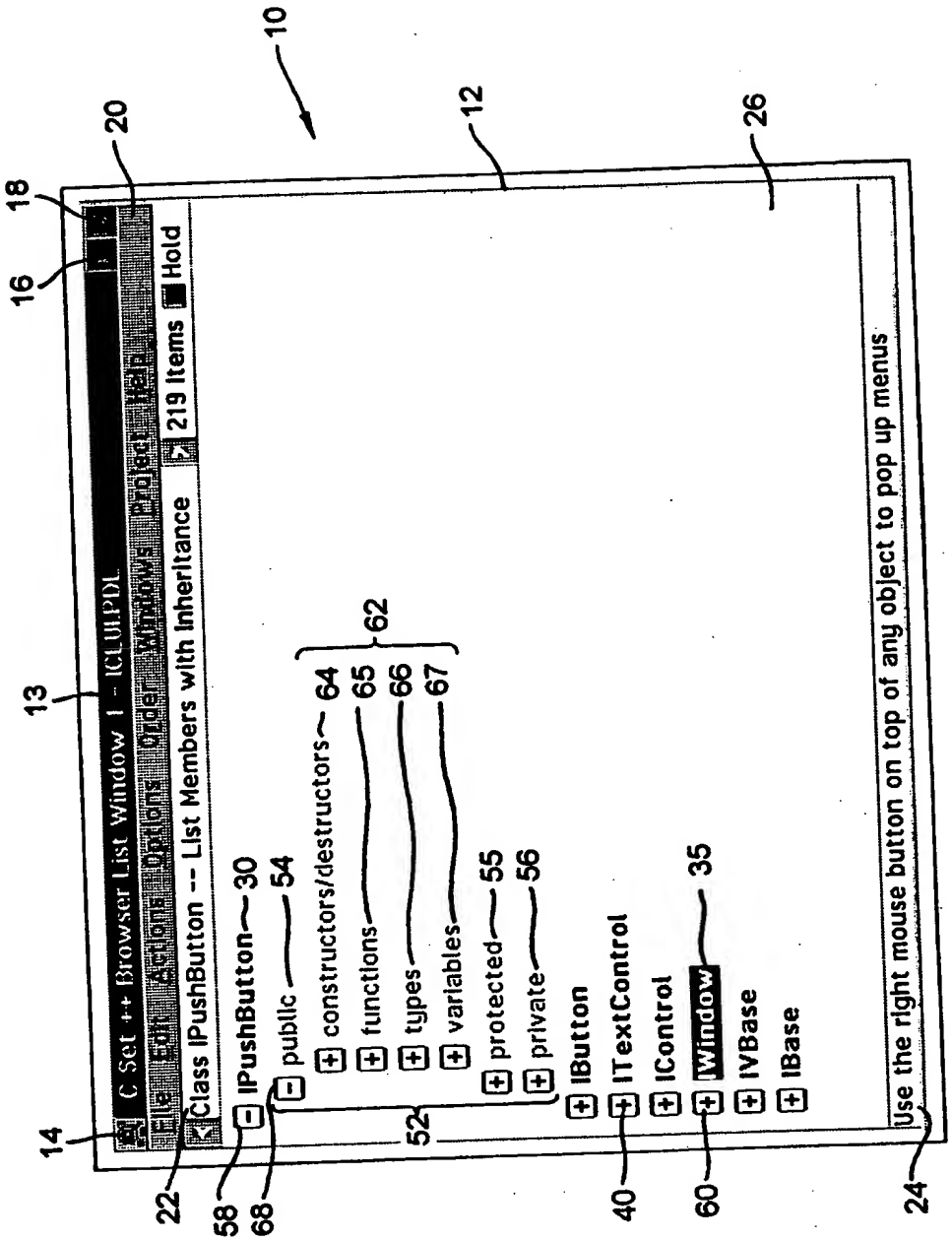

FIG. 4

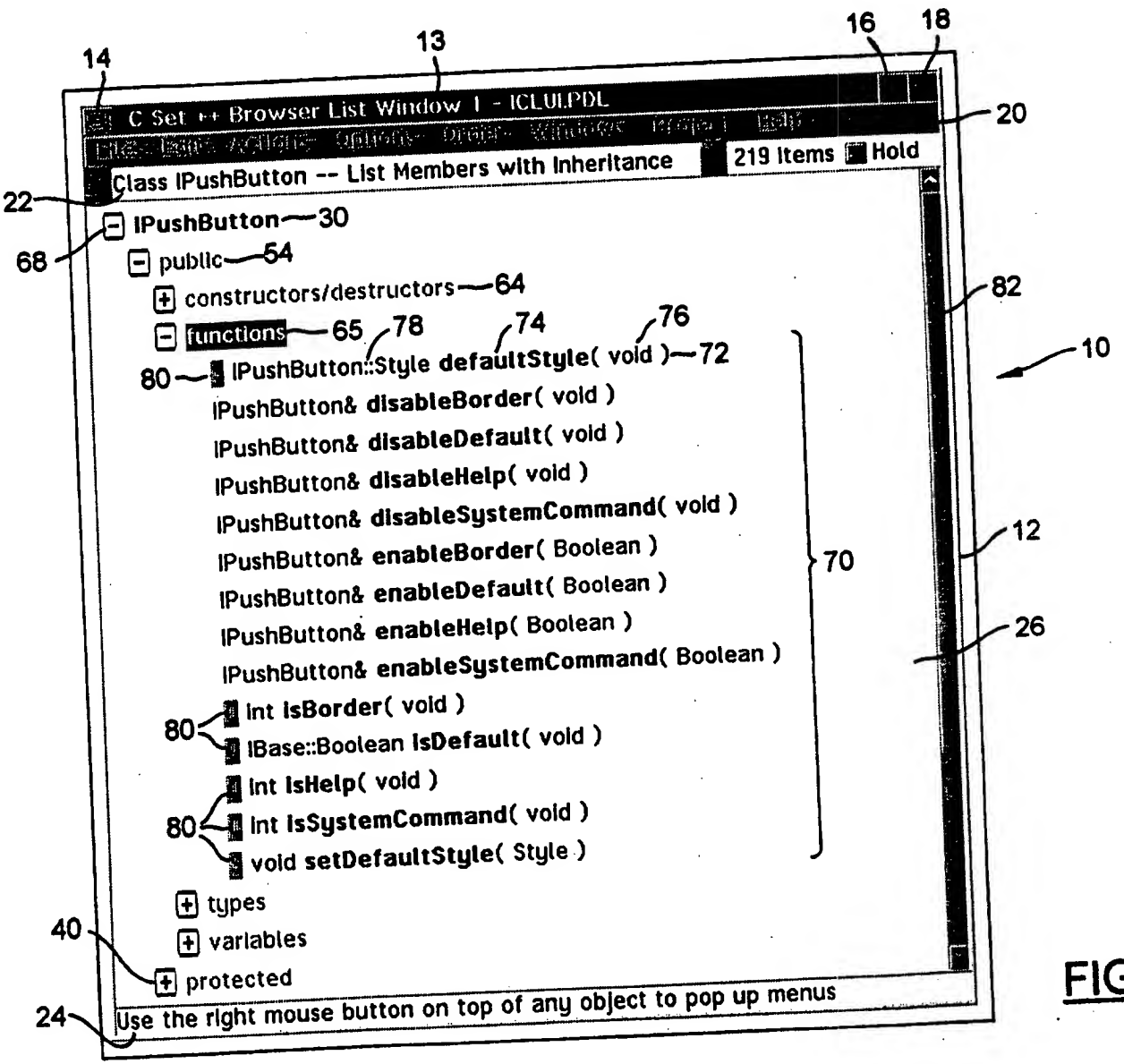

FIG. 5

2136155.

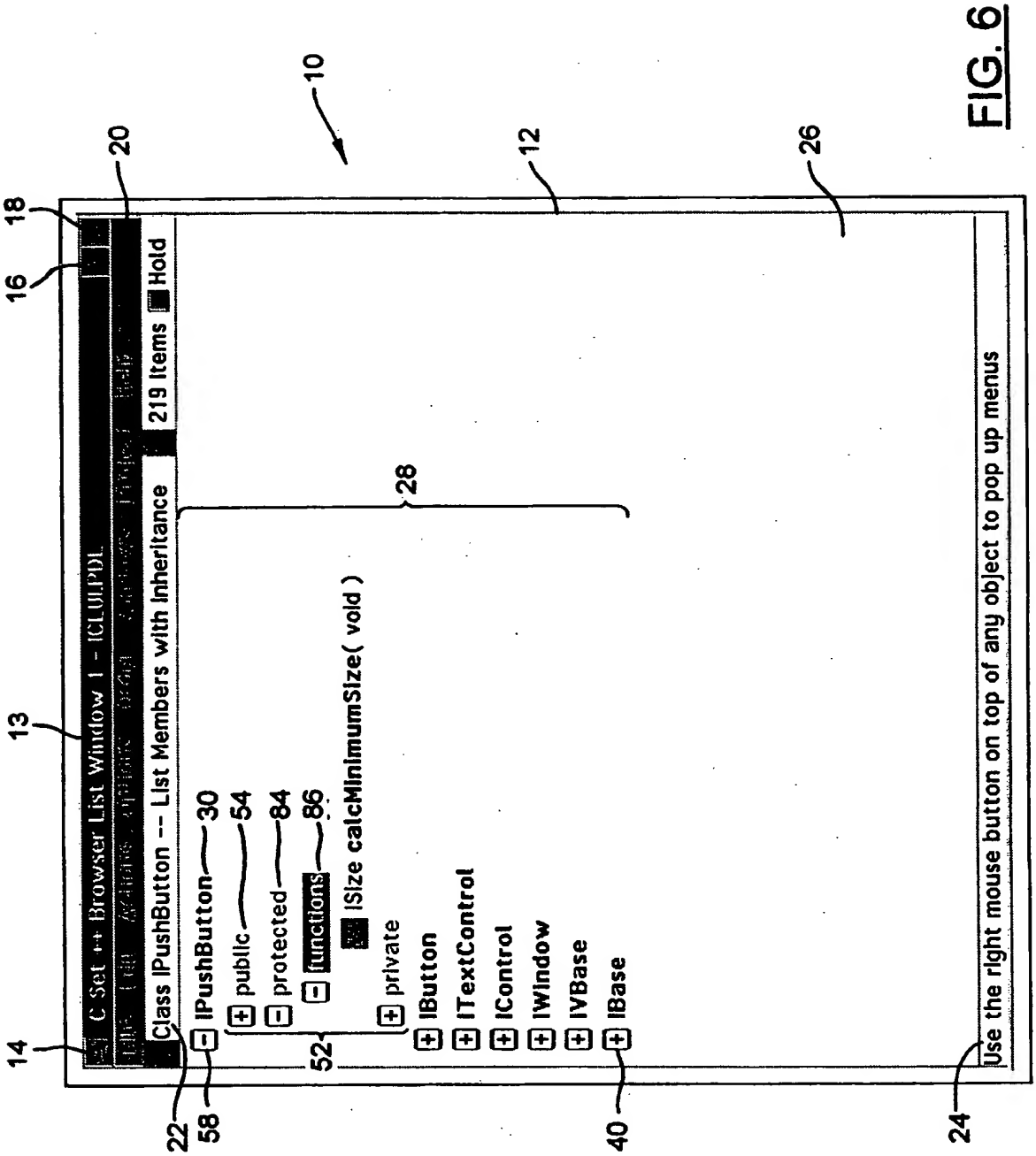

FIG. 6

2136155

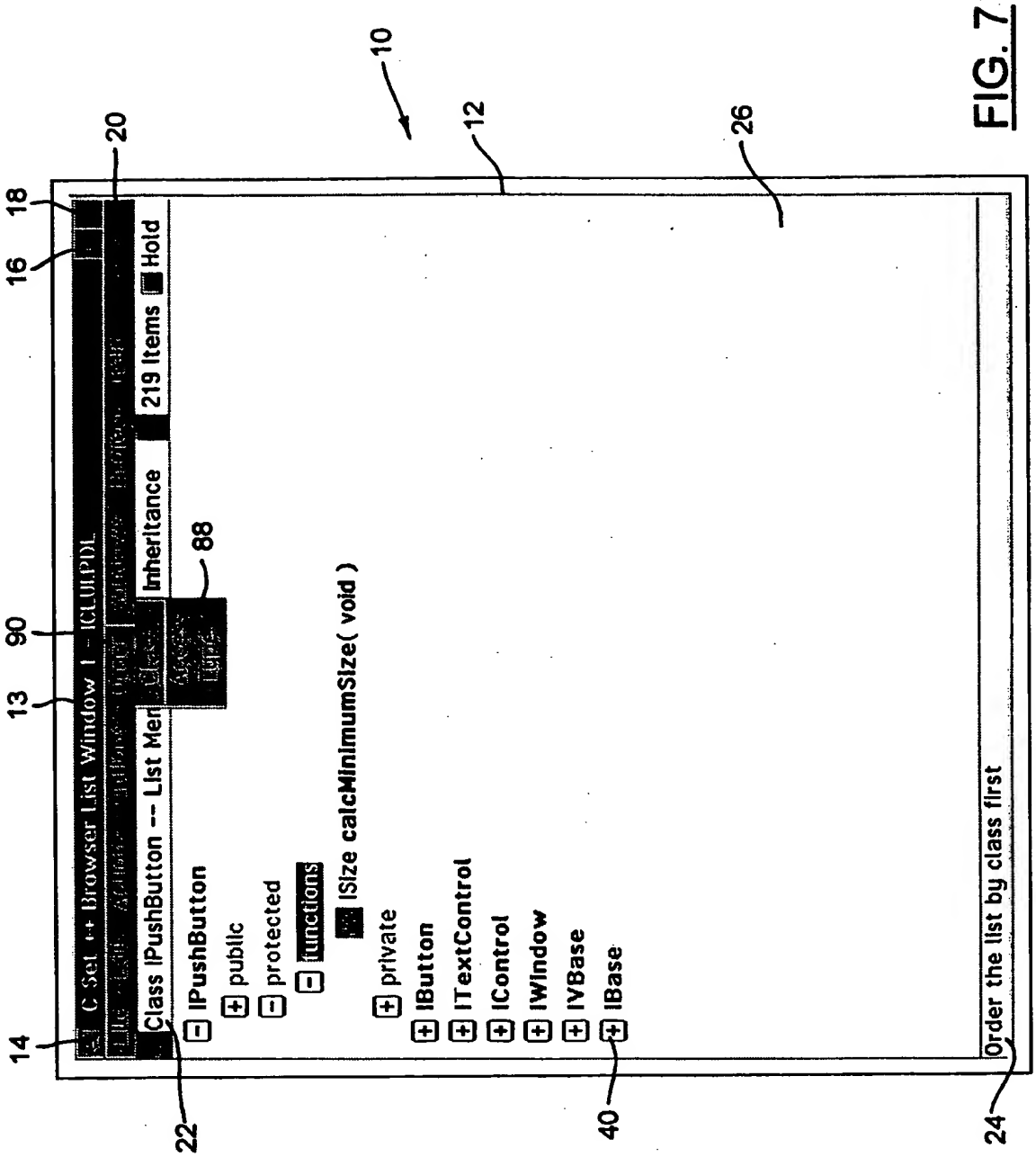

FIG. 7

2136155

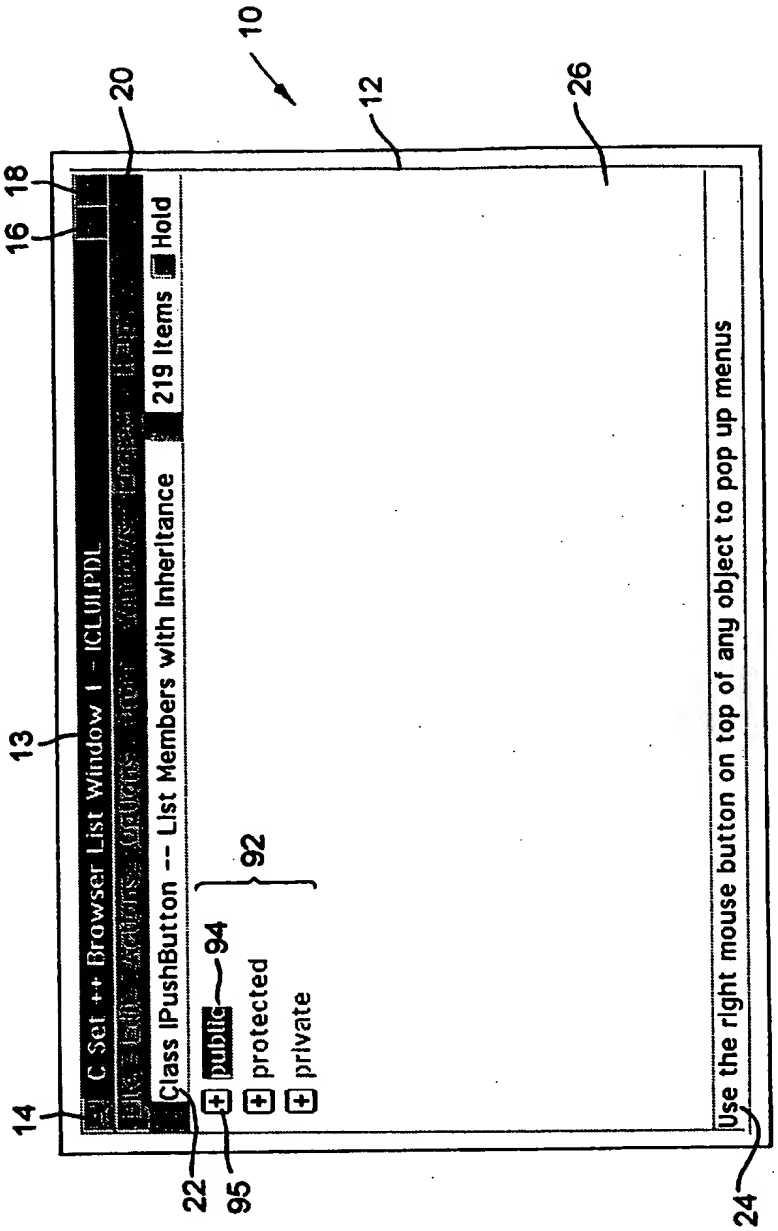

FIG. 8

2136155.

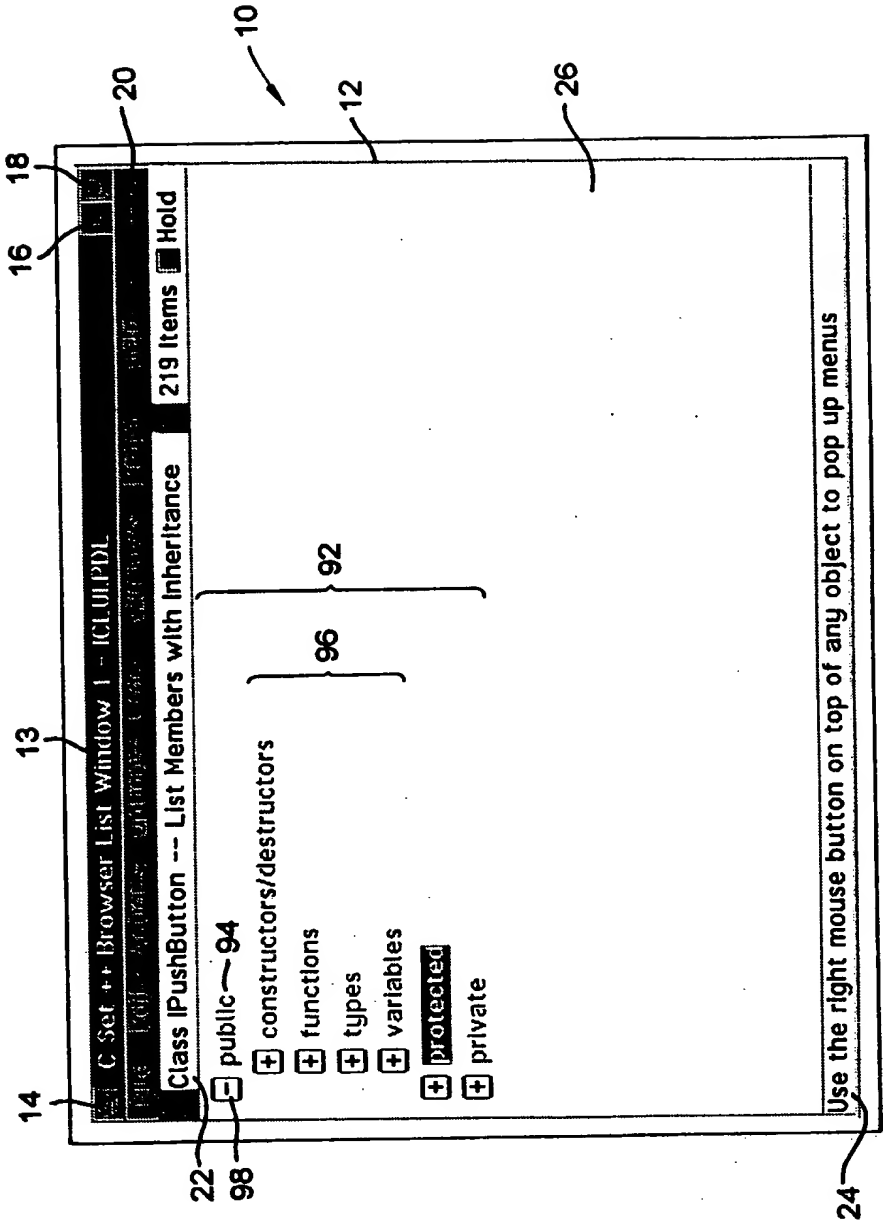

FIG. 9

2136155.

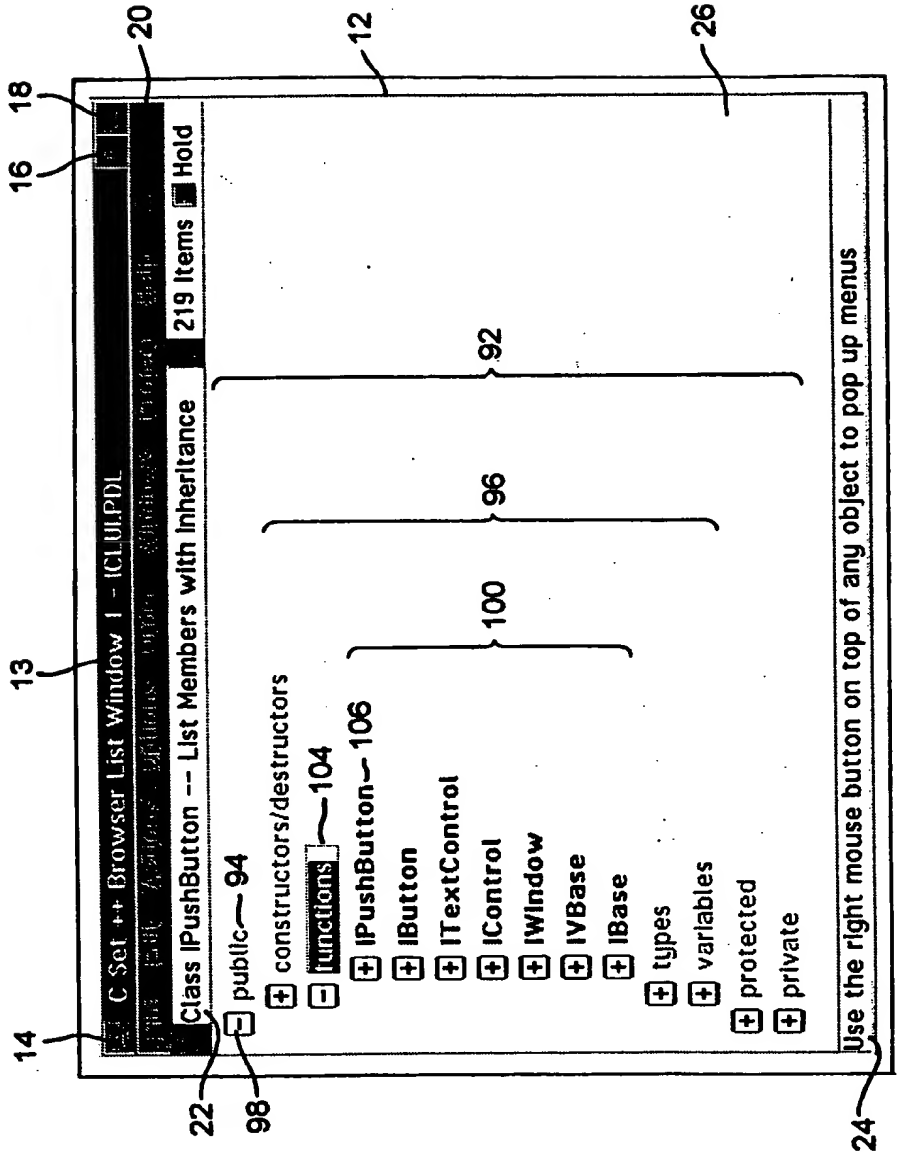

FIG. 10

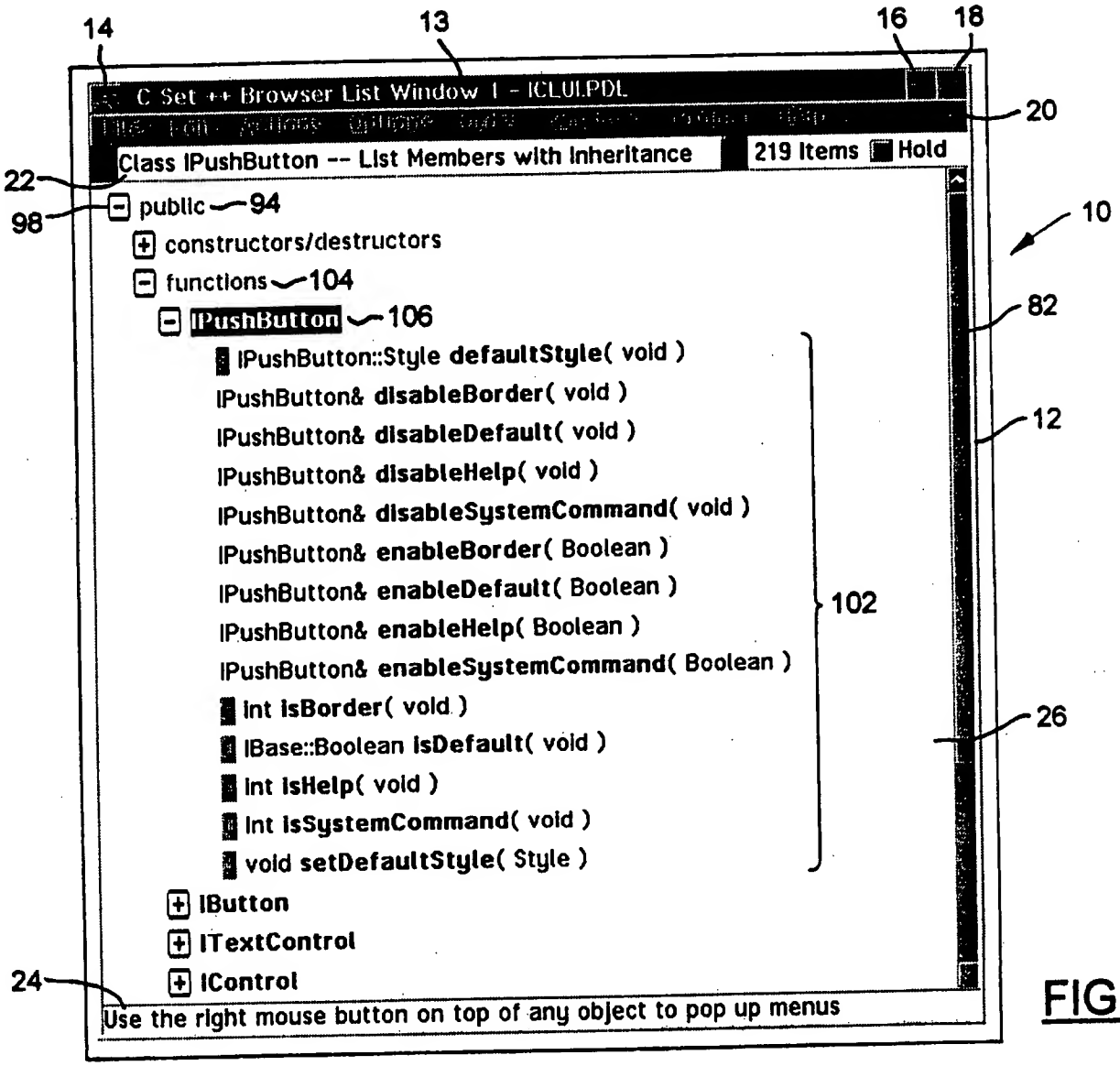

FIG. 11

2136155.

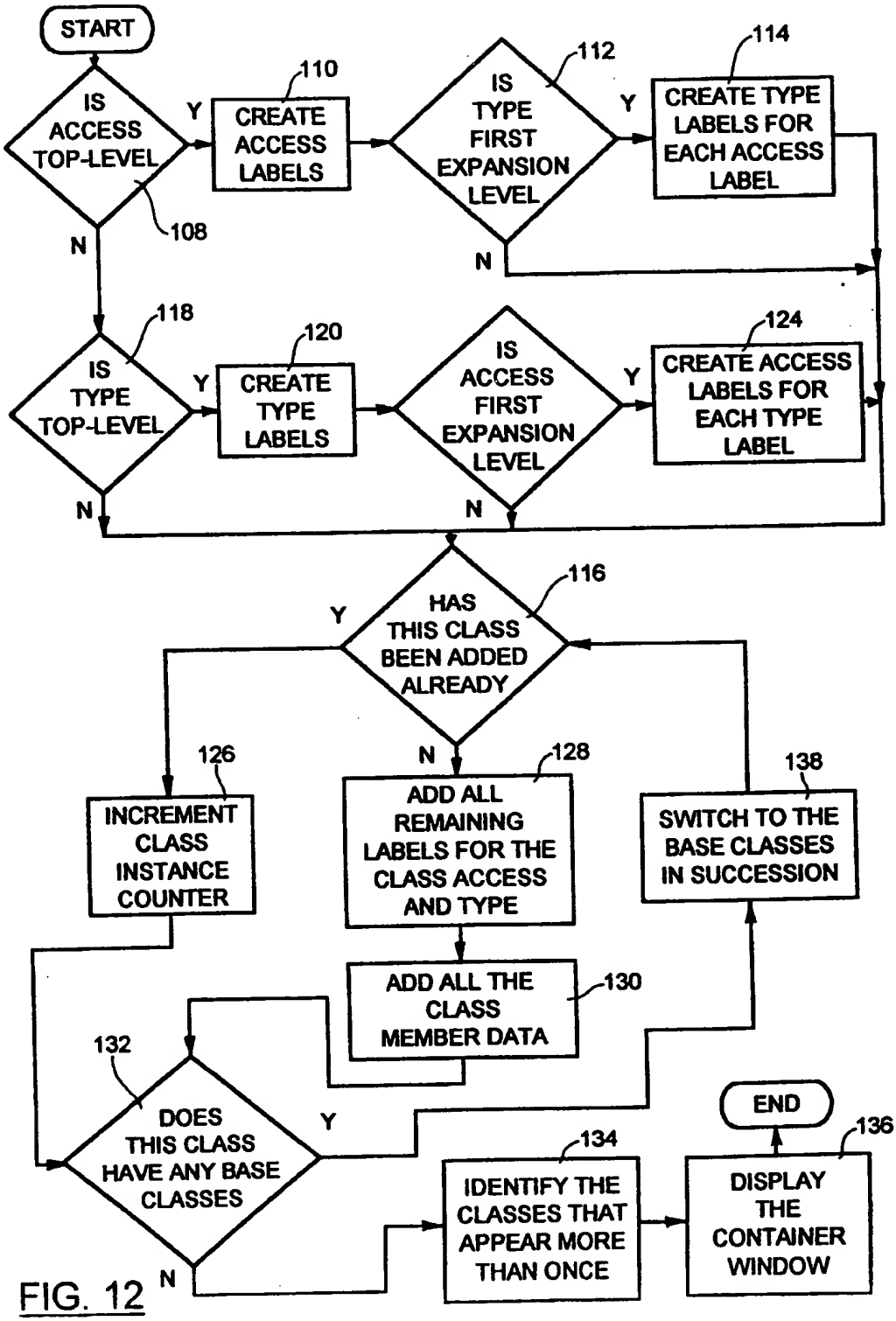

FIG. 12

2136155.

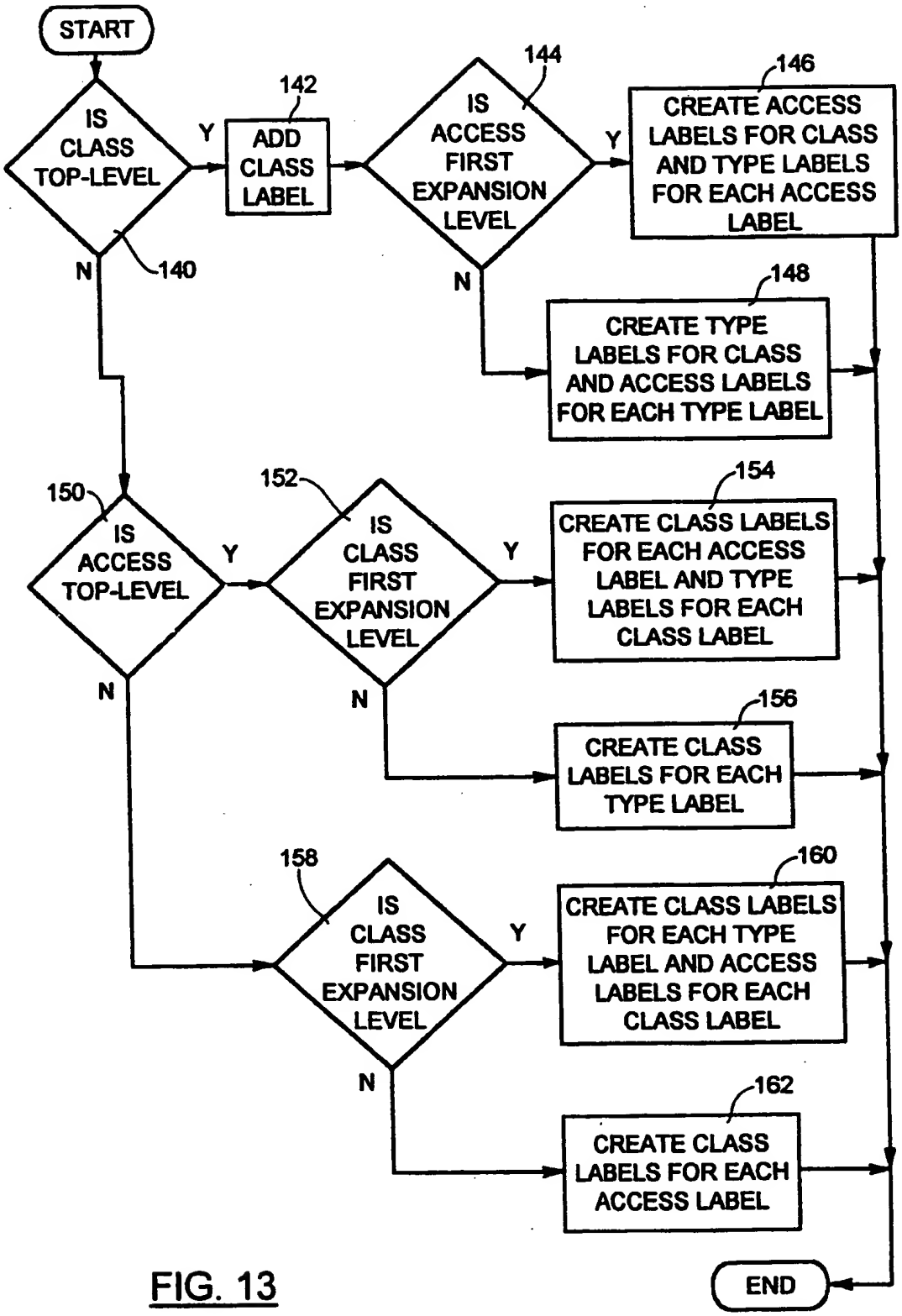

FIG. 13

2136155.

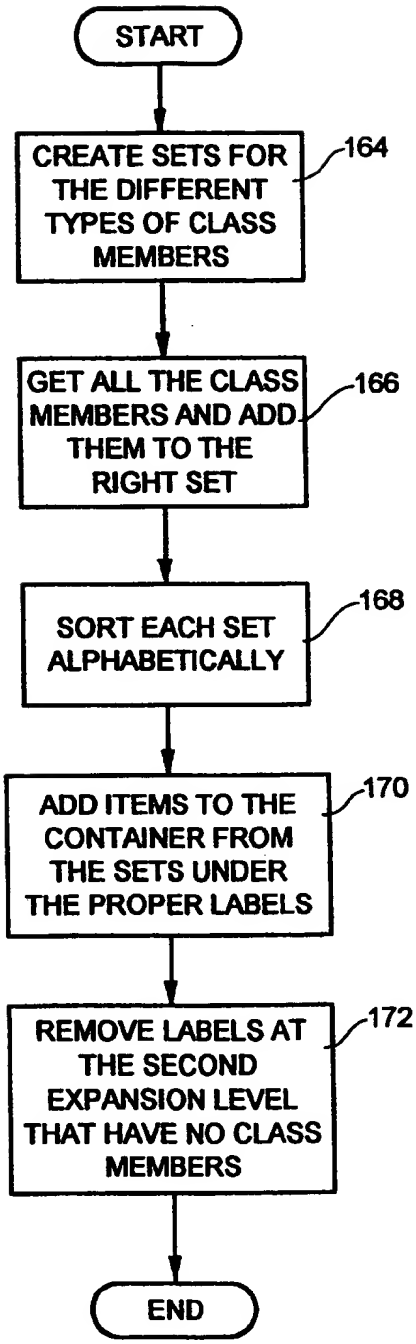

FIG. 14